

BACKGROUND TO THE INVENTION

Technical Field

10

15

The advantages of such a publish/subscribe, content-based addressing systems are well known and include the ability to decouple subscribers and publishers from one another. This decoupling allows publishers and subscribers to operate without having any knowledge of the identity, location or address, or communication protocols of each other. The flexibility that this offers is enormous and, accordingly, such content/subject-based addressing communication-environments are becoming increasingly popular.

Unfortunately, the very advantages (such as anonymous decoupling) of these systems, precludes the use of conventional reliable messaging protocols such as TCP. TCP, and other reliable messaging protocols apply only in point-to-point type of communications. In these point-to-point communications message senders and

receivers are directly linked to one another and therefore know each other's addresses and locations.

- 5 Unfortunately, these reliable messaging protocols -- that guarantee arrival and order of arrival of messages -- require advance knowledge between applications. They are, therefore, not applicable to typical publish/subscribe environments.

- 10 Yet, such reliable or certified delivery of messages is extremely important. For example, certified delivery is appropriate when a sending application requires individual confirmation of delivery for each message it sends. For example, a travelling sales representative computes sales figures on a lap-top computer, and sends them to a supervisor at the office. The user must know for certain that the data has arrived, and has been included in the supervisor's sales report.

- 15 Certified delivery is also appropriate when a receiving application cannot afford to miss any messages. For example, in an application that processes orders to buy and sell inventory items, each order is important. If any orders are omitted, then inventory records are incorrect.

- 20 In addition, certified delivery is appropriate when each message on a specific subject builds upon information in the previous message with that subject. For example, a sending program updates a receiving database, contributing part of the data in a record, but leaving other parts of the data unchanged. The database is correct only if all updates arrive in the order they are sent.

- 25 Furthermore, certified delivery is appropriate in situations of intermittent physical connectivity--such as discontinuous network connections, for example, an application in which several mobile lap-top computers must communicate with one another. Connectivity between mobile units is sporadic, requiring persistent storage of messages
30 until the appropriate connections are reestablished.

Thus, a very real need exists for having both the advantages of certified messaging and the advantages of content-based, anonymous publish/subscribe environments.

SUMMARY OF THE INVENTION

Briefly, according to this invention a publisher publishes a message to any number of unknown subscribers. Note, as used herein "publisher" and "sender" are used synonymously, and "subscriber" and "listener" are used synonymously. The message is published indicating the subject or content using typical content-based publish/subscribe protocols. Subscribers interested in receiving information on the designated content receive the message without knowing about the publisher. Thus the publisher information remains transparent to the subscriber.

In circumstances where the certified messaging is required, the invention provides for establishing a message delivery tracking session. This session includes a name and a ledger used for tracking.

Using these functions the system can track delivery of messages and notify publishers/senders if messages are not delivered. In one embodiment of the invention, delivery attempts are repeated for a preset time (or number of delivery attempts) to ensure or attempt to ensure delivery.

This invention also extends to queuing messages for certified delivery. This occurs when certified delivery is required for a one of a group of n possible recipients. The system ensures that one (and not all) of the group receives the message. This is accomplished by having members of the group indicate their availability or capacity and having the system route the message to the subscriber (listener) with the greatest availability.

An extension of this concept is the scheduling of tasks for a group of n possible task performers, each available to accomplish a task. Each task doer, notifies the system of its availability/ability to accomplish tasks. Tasks are then sent to/queued for each task doer according to a rating based on its availability.

Advantages of the Invention

This invention has a number of advantages. For example, it provides:

Certainty

Certified delivery assures application programs that every message reaches each

intended recipient--in the order sent. When delivery is not possible, both senders and, optionally, listeners receive explicit information about each undelivered message.

Convenience

5 Once a program sends a certified message, the system continues delivery attempts until delivery succeeds, or until the message's time limit expires.

Control

Application programs determine an explicit time limit for each message.

10 Sending applications can disallow certified delivery to specific listening sessions.

Detail

The system can also present advisory messages to inform application programs of every significant event relating to delivery.

Process-Based or File-Based Recording

15 The system can also record the status of each message in a ledger. Applications that require certification only for the duration of the application

process can choose a process-based ledger. Applications that require certification that

20 transcends process termination and restart can choose a file-based ledger.

The invention will be described in greater detail below with reference to the accompanying drawings.

25 DESCRIPTION OF THE DRAWINGS

In the attached drawings:

Figure 1 is a schematic representation of a typical publish/subscribe environment useful for illustrating this invention; and

Figure 2 is a schematic representation of a typical publish/subscribe environment useful in illustrating the distributed queuing and task scheduling aspects of the invention.

SPECIFIC DESCRIPTION

OVERVIEW

Figure 1 shows a publisher application (sender) 10 and a plurality of subscriber applications (listeners) 20, 20' and 20". In the preferred embodiment of this invention the publisher and subscriber(s) are software applications based on one or more computers interconnected by a network 30 providing a data path among the applications. The publisher 10 and subscriber(s) 20 preferably implement a content-based communications protocol whereby a publisher publishes a message indicating only the content of the message and without knowing the identity or protocols used by the subscriber(s) 20. These inter-application communications are established by communications daemons 12 (associated with a publisher/sender) and 22, 22' and 22" (associated with the subscriber/listener 20, 20' and 20"). The arrangement shown in this figure is well known and described in many publications including the patents referred to above.

As is described in much greater detail below, a listener 20 can register with a specific publisher 10 to receive certified messages. This communication includes the subscribers name, its "inbox" address and the subject/content of messages it requires information on. Thus the publisher 10 will have a list of subscriber names and inboxes (but know nothing else about the subscriber) for all subscribers wishing to receive certified messages. The publisher/sender 10 will therefore expect an acknowledgement of each message it sends out; an acknowledgement it would receive from a subscriber/listener 20, 20' and/or 20". Importantly, if the publisher/sender 10 does not receive the acknowledgement, it sends an acknowledgement request message, usually for a predetermined time or number of "sends."

In the event the subscriber wishes to have guaranteed delivery of messages, the publisher can save the message to disk (or other storage) until an acknowledgement of subscriber receipt occurs. Thus, until message times out, the subscriber can, at a later date, receive the message by contacting the publisher. This would usually happen where messages are very dependent on their sequence or build upon prior message. In these circumstances missing/unreceived messages could be catastrophic. Also, in these (and in most certified messaging applications of this invention) each certified message is assigned a tracking number. This allows both sender and listener/subscriber to monitor which messages are received and/or missing.

In many applications, data communications are highly reliable, but in some situations applications require even stronger assurances of delivery. Certified delivery features offers greater certainty of delivery even in situations where processes and their network connections are unstable.

5

CERTIFIED MESSAGING

Enabling a Delivery-Tracking Session

10 The first step toward certified delivery is to enable a delivery-tracking session. A delivery-tracking session begins as an ordinary anonymous publish/subscribe session; enabling a session adds information so that it can participate in certified delivery protocols. The additional information includes a name and a ledger.

15 Delivery-tracking sessions can send and receive messages, just as ordinary sessions can. In addition, delivery-tracking sessions can participate in certified delivery calls (that is, calls in the rvcn library layer); ordinary sessions cannot participate in these calls. (Notice the asymmetry. Delivery-tracking sessions can participate in ordinary calls, but ordinary sessions cannot participate in certified delivery calls.)

20 *Name*

Each delivery-tracking session has a name which may be reusable, or non-reusable. The name identifies the session to other delivery-tracking sessions, and is part of the label that identifies outbound messages from the session.

25 A name is reusable when a program supplies it explicitly to the enabling call. When a session with a reusable name also has a file-based ledger, it operates as an instance of a persistent correspondent--which allows continuity of certified delivery beyond session termination and program restarts.

30 Two delivery-tracking sessions must not bind the same reusable name that is, at any moment in time, each reusable name must be unique. Sessions may reuse a name sequentially, but not simultaneously. Violating this rule can significantly obstruct certified delivery. Typically, session names have the same syntax as subject names.

Programs may omit a name in the enabling call--in which case the call generates a unique, non-reusable name for the session. No other session on any computer can ever have the same name. As a result, a session with a non-reusable name operates as a transient correspondent--no subsequent session can continue the certified delivery behavior of the session.

Enabling a delivery-tracking session creates a ledger for it. Certified delivery software uses the ledger to record information about every unresolved outbound certified message, every subject for which this session receives (inbound) certified messages, and other cooperating delivery-tracking sessions.

Programs may store the ledger in a ledger file, or in process-based storage within the running program. (Even when a session uses a ledger file, it may sometimes replicate parts of the ledger in process-based storage for efficiency; however, programmers cannot rely on this replication.)

Ledger files must be unique. That is, two sessions must not use the same ledger file (concurrently). If an operating system supports raw disk devices (also called raw partitions), a device can be specified as the ledger file.

A session with a file-based ledger and a reusable name qualifies as a persistent correspondent, with certified delivery behavior that can extend beyond session termination.

Labeled Messages

A labeled message is like an ordinary message, except that it includes supplementary information, which delivery-tracking sessions can use for certified message delivery:

The name of the delivery-tracking session that sent the message.

A sequence number assigned by the sending session.

Sending a Labeled Message

delivery agreement, the participants are delivery-tracking senders and listeners; afterward, they are certified senders and listeners. A labeled message is only a certified message when the sender and receiver maintain a certified delivery agreement.

5 DELIVERING A CERTIFIED MESSAGE

Once a delivery agreement is in place, all subsequent messages on the subject (from the certified sender to the certified listener) are certified messages. Each **certified message** generates a series of protocol events:

10 When the system presents a certified message to the listening callback function, it includes the sequence number assigned (automatically) by the sending Software and the publisher's name.

15 When the callback function returns, certified delivery software automatically confirms delivery to the sender and records confirmation to a ledger. (Programs can override this behavior and confirm delivery explicitly.)

20 When confirmation reaches the sending program, certified delivery software records delivery in the sender's ledger, and presents a **DELIVERY.CONFIRM** advisory to the sender session.

25 When confirmation has arrived from every certified listener for this message, certified delivery software deletes the message from the sender's ledger, and presents a **DELIVERY.COMPLETE** advisory to the sender session.

Automatic Confirmation of Delivery

The default behavior of certified listeners is to automatically confirm message delivery upon return from the data callback function. Programs can selectively override this behavior for specific listening endpoints (without affecting other listening endpoints).

30 By overriding automatic confirmation, the listener assumes responsibility for explicitly confirming each inbound certified message.

Consider overriding automatic confirmation when processing inbound messages involves asynchronous activity, such as computations in other threads, database queries, or additional network communications.

5 ***Requesting Confirmation***

If a certified sender does not receive prompt confirmation of delivery from a certified listener (for example, because of network glitches), the system in the sending program automatically requests confirmation. After each request, it presents a DELIVERY.NO_RESPONSE advisory to the sending session.

10

When a listener receives a request for confirmation, it checks its ledger, and reconfirms receipt of the messages that it has already confirmed. (This behavior is identical, whether the program uses automatic confirmation, or overrides it.)

15 **SEQUENCING AND RETRANSMISSION**

A delivery-tracking sender assigns sequence numbers serially for each outbound subject, so the sequence numbers reflect the order of messages from a sender on a specific subject.

20 When certified delivery software presents certified messages to a certified listener, it preserves the sequence in which the sender sent them. If a message arrives out of sequence, certified delivery software in the listener does not present it to the callback function until all the preceding messages are available.

25 For example, a certified listener is receiving certified delivery for the subject FOO from a sender named BAZ. After receiving and presenting message number 32, the next message to arrive is message 35. Certified delivery software holds message 35 until it can first present messages 33 and 34.

30 Meanwhile, the certified delivery software in FOO requests retransmission of messages 33 and 34 from Baz. In a case where the time limit on those messages has expired--so BAZ no longer has them in its ledger---certified delivery software presents a DELIVERY.UNAVAILABLE advisory to the listener, indicating that messages 33 and 34 are no longer available. Then it presents message 35 to the data callback function.

PERSISTENT CORRESPONDENTS

We introduced the concept of persistent correspondents in the section Name, page 144.

- 5 A reusable name and a file-based ledger allow a persistent correspondent to continue certified delivery beyond the termination and restart of a session or process.

Example

- 10 Consider an example application system, in which application program JOE generates important information, and sends it to application program SUE in certified messages on the subject REMEMBER.THIS. Upon receipt, SUE stores the information in a database.

- 15 If either JOE or SUE terminate unexpectedly, it is crucial that certified messages still arrive for entry into the database. To ensure this result, both programs must represent persistent correspondents—that is, both programs enable sessions with reusable names (JOE_PC and SUE_PC), and each program keeps a file-based ledger. In addition, SUE requires old messages when it enables the session SUE_PC.

- 20 During operation, JOE has sent message number 57 on the subject REMEMBER.THIS but has not yet received delivery confirmation for messages 53- 56. SUE is processing message 53, when a sudden hardware failure causes to terminate. Meanwhile, JOE continues to send messages 58-77.

- 25 The computer restarts, and SUE restarts. The ledger file for SUE_PC indicates that message 52 was received and confirmed for the subject REMEMBER.THIS for a given publisher as JOE_PC. On restart SUE_PC will contact JOE_PC and reestablish their certified delivery agreement. When JOE accepts, JOE_PC retransmits the stored messages 53-77 on that subject.

- 30 In the above scenario it is important to notice the following:

That SUE does not miss any REMEMBER.THIS messages. However, the new SUE must gracefully fix any difficulties caused by partial processing of message 53 by the old SUE.

JOE and SUE communicate using a broadcast subject name not an inbox. Inbox names are unique, so they cannot continue beyond session termination and restart.

5 ANTICIPATING A LISTENER

In some situations, a delivery-tracking sender can anticipate the request for certified delivery from a persistent correspondent that has not yet begun listening.

10 Consider an example in which a database program (DB) records all messages with the subject STORE.THIS. The program DB enables a session that instantiates a persistent correspondent named DB_PC. All programs that send messages with the subject STORE.THIS depend on this storage mechanism.

15 One such sending program is JAN. Whenever JAN starts, it can anticipate that DB_PC will request certified delivery of the subject STORE.THIS. Suppose that JAN starts, but DB is not running, or a network disconnect has isolated JAN from DB. Anticipating that it will eventually receive a registration request for STORE.THIS from DB_PC, JAN makes an add listener call. The effect is that the software within JAN behaves as if it has a certified delivery agreement with DB_PC for the subject STORE.THIS. It stores outbound
20 messages (on that subject) in its ledger. When DB restarts, or the network reconnects, JAN automatically retransmits all the stored messages to DB.

CANCELLING CERTIFIED DELIVERY

Both listeners and senders can cancel a certified delivery agreement.

25

Listeners cancel by closing the listening endpoint, using calls listed below. Senders with certified delivery agreements to the closed endpoint receive REGISTRATION.CLOSED advisories. HOST.LISTEN.STOP advisories inform other applications of the change.

30 Senders can cancel certified delivery of a specific subject to a specific listener. The sender program deletes from its ledger all information about delivery of the subject to the listener. The sending program receives a REGISTRATION.CLOSED advisory. If the listening correspondent is available (running and reachable), it receives a

REGISTRATION.NOT_CERTIFIED advisory. (Unlike the disallow listener calls in Table 7, these calls do not cause denial of subsequent registration requests.)

DISALLOWING CERTIFIED DELIVERY

- 5 As described before senders automatically accept all registration requests. This is true except when the sending program explicitly disallows certified delivery to a listening session. Calls that disallow a listener cancel existing certified delivery agreements with the listener session (on all subjects), and cause certified delivery software to automatically deny subsequent registration requests from the listener session.

10

When a sender has disallowed a listener, the events connected with registration do not occur. Instead, certified delivery software in the sender notifies the listener session that the request is disallowed. When certified delivery software in the listening program receives the rejection notice, it presents a **REGISTRATION.NOT_CERTIFIED** advisory to the

15

Allow listener calls supersede the effect of a previous disallow listener call, allowing subsequent registration requests from the listener session to succeed.

20 NO RESPONSE TO REGISTRATION REQUESTS

It is possible that a registration request never reaches the delivery-tracking sender, or the acceptance notice never reaches the listening program (for example, because of network glitches, or termination of the sending program). After repeated attempts to register without response from the sender, certified delivery software in the listening

25

program presents a **REGISTRATION.NO_RESPONSE** advisory to the listening session. After several attempts to register with no response, the listener stops sending requests.

REUSABLE NAMES

- 30 Sessions that represent persistent correspondents require reusable names. Reusable names must obey the syntax for subject names. Reusable names must not contain wildcard characters. Reusable names may not begin with reserved elements (such as **_INBOX**, **_RV** or **_LOCAL**). For best performance, reusable names should be kept short--only a few characters, typically no more than three or four elements, and no more than 50 characters in total.

LEDGER STORAGE MODE

- Each delivery-tracking session records information in a ledger, which occupies storage space within the application process. A session that represents a persistent correspondent must also keep a copy of the ledger in a file. The file-based ledger preserves certified delivery information beyond session (or process) termination and restart.

This feature has two associated costs:

10

The ledger file consumes disk space.

The application program pauses to update the ledger file at each significant event.

15

Transient correspondents need not pay these costs, because they do not use ledger files. However, keeping the ledger in process-based storage consumes process memory.

Ledger Size

- 20 The size of the ledger depends on several factors---the most important of which is the retention rate of stored data. That is, the ledger grows fastest in response to the cumulative length of incompletely delivered messages.

- 25 Program developers can estimate the expected size of the ledger, and must ensure that the process can allocate sufficient memory to contain it. For a file-based ledger, ensure that sufficient disk space is available as well, as memory requirements for the process application change when utilizing a file-based ledger.

EVENT MANAGER

- 30 The system's certified delivery depends on the event manager for timer and I/O events. When an application enables a delivery-tracking session, that session must be an event-managed session.

No Synchronous Sessions

Queue Members

The member sessions of a distributed queue all share the same reusable correspondent name indicating that they are members of the queue with that name. Each member of a distributed queue listens for the same subjects--yet even when n members listen, for each inbound message (or task), only one member processes the message.

Member Roles--Worker and Scheduler

As used herein with respect to queueing and queued delivery, the terms "listener" and "worker" are used interchangeably. Each distributed queue member session can have two distinct roles--as a worker, and as a potential scheduler.

In the **listener or worker role**, queue member sessions support a limited subset of certified delivery calls; members can listen to a subject, override automatic confirmation of delivery and confirm delivery. Queue member sessions do not support any other certified delivery calls (in particular, calls associated with sending certified messages). However, they do support all standard calls (for example, sending ordinary messages).

The system includes fault tolerance software that maintains exactly one active scheduler in each queue; if the scheduler process terminates, another member assumes the role of scheduler. The queue member session in the **scheduler role** assigns inbound tasks to listeners in the queue. (A scheduler can assign tasks to its own listener component, but only does so when all other listeners are busy.)

The Scheduler as a Fault-Tolerant Component

Although any queue member has the potential to become the scheduler, fault tolerance software maintains exactly one scheduler at all times. Fault tolerance parameters guide the software to select the most suited member as scheduler.

Scheduler weight represents the ability of a member session to fulfill the role of scheduler, relative to other members of the same queue, i.e., the greatest availability or unused member resources. The queue members use relative scheduler weight values to

Task Capacity

Task capacity is the maximum number of tasks that a worker or listener can accept. When the number of accepted tasks reaches this maximum, the worker or listener cannot accept additional tasks until it completes one or more of them.

When the scheduler receives a task, it assigns the task to the worker or listener with the greatest worker or listener weight--unless the pending tasks assigned to that worker or listener exceed its task capacity. When the preferred worker or listener has too many tasks, the scheduler assigns the new inbound task to the worker or listener with the next greatest worker or listener weight.

Enabling a session as a queue member tacitly sets its worker or listener task capacity to

1. Programmers can tune task capacity based on two factors:

Multi-tasking program on multiprocessing hardware.

On a multiprocessing computer, a multi-threaded program that devotes n threads and n processors to inbound tasks has task capacity n .

Communication time lag.

In most distributed queue applications, the communication time is an insignificant fraction of the task turnaround time. That is, the time required to assign a task and signal its completion is very small compared to the time required to process the task itself. For example, when average task turnaround time is 2000 milliseconds, of which communication time contributes only 10 milliseconds to the total, then task capacity is the same as the number of processors or threads.

However, in some situations communication time can be significant—for example, when the queue members are distributed at distant sites connected by a WAN. When communication time is significant, the meaning of task capacity changes; instead of signifying the number of tasks that a listener can process concurrently, it signifies the

number of tasks that can fill the listener's capacity despite the communication time lag. For example, when the average task turnaround time is 1500 milliseconds, of which the average task processing time contributes 1000 milliseconds to the total, then setting the task capacity to 3 minimizes the listener's idle time between tasks.

- 5 When tuning task capacity to compensate for communication time lag, balance is critical. Underloading a listener (by setting its tasks capacity too low) can cause the listener to remain idle while it waits for the schedule to assign its next task. Conversely, overloading a listener (by setting its task capacity too high) can cause some assigned tasks to wait, while other listeners that might have accepted those tasks remain idle.

10

TASK SCHEDULING

- In a further application of this invention, the broad concepts of distributed queuing can be applied to scheduling tasks for different task doing applications. In this application an application can be either a scheduler or a worker or both. Each worker is assigned a weight indicating its ability to do work, usually the number of tasks it can do simultaneously. Typically, workers assign their own weights.
- 15

- One application/member of a group becomes the scheduler. Once this occurs, all other applications become designated workers only. The scheduler becomes both a worker and the scheduler.
- 20

- When a task is received by the group (the scheduler) it assigns the task to the worker with the highest "weight" or ability to do tasks. The worker then calls back to the scheduler accepting the task and, upon completion, returns a call to the scheduler indicating this fact. In certain instances, the call back to the scheduler includes both an acceptance and a notification of task completion.
- 25

- Two Appendices, A and B are attached. These appendices include detailed C coding information respectively for Certified messaging and for message queuing.
- 30

All publications and patent applications mentioned in this specification are herein incorporated by reference to the same extent as if each individual publication or patent application was specifically and individually indicated to be incorporated by reference.

The invention now being fully described, it will be apparent to one of ordinary skill in the art that many changes and modifications can be made thereto without departing from its spirit or scope.

THE BIBLE

Appendix A: Certified Message Delivery (Programming Details for C Programmers)

5 As indicated previously, even though some communications are highly reliable, certain applications require even stronger assurances of delivery. Certified delivery features offers greater certainty of delivery even in situations where processes and their network connections are unstable.

10 This Appendix A provides programming details for C programmers wishing to implement Certified message Delivery. The Appendix provides, in Table A1, an overview listing of Certified's messaging Deliver Datatypes and Functions. Each Datatype or Function is then described in greater detail with Cross-references to related Datatypes or Functions.

15

Certified Message Delivery CAPI

The following table summarizes the datatypes and functions in the certified message delivery C API.

20

Table A1: Certified Message Delivery: Datatypes and Functions

Item	Description
<code>rvcm_Enable ()</code>	Enable an existing session for delivery tracking.
<code>rvcm_ListenSubject ()</code>	Listen for broadcast messages, and request certified delivery whenever available.
<code>rvcm_ListenInbox ()</code>	Open a delivery-tracking inbox to listen for point-to-point messages, and request certified delivery whenever available.
<code>rvcm_Callback</code>	Function type of callback functions that receive and process inbound messages for delivery-tracking listeners.
<code>rvcm_ListenId</code>	Certified listening calls return identifiers of this type.
<code>rvcm_seq</code>	Certified messages bear sequence numbers of this type.
<code>rvcm_close ()</code>	Close a delivery-tracking endpoint; stop listening for messages on it.
<code>rvcm_Send ()</code>	Send a labeled message, and track delivery to cooperating listeners.
<code>rvcm_SendWithReply ()</code>	Send a labeled request message, and track delivery to cooperating listeners.
<code>rvcm_AddListener ()</code>	Pre-register an anticipated listener.

Table A1: Certified Message Delivery: Datatypes and Functions (con't.)

Item	Description
rvcm_RemoveListener()	Cancel certified delivery of a subject to a listening correspondent.
rvcm_DisallowListener()	Cancel certified delivery of all subjects to a listening correspondent, and deny subsequent registration requests.
rvcm_AllowListener()	Invite a receiver to reinstate certified delivery for its listeners.
rvcm_NoAutoConfirm()	Override automatic confirmation of delivery.
rvcm_confirm()	Confirm delivery of a certified message.
rvcm_ReviewLedger()	Summarize the delivery status of messages in the ledger,
rvcm_ReviewCallback	Function type of callback functions that process ledger information for rvcm_ReviewLedger() .
rvcm_SyncLedgerFile()	Synchronize the ledger file.
rvcm_Error	Datatype. Enumerates error codes for the certified message delivery API.
rvcm_ErrorText()	Return a text string describing an error code.

rvcm_Enable()

Declaration

```

5   rvcm_Error rvcm_Enable (
      rv_Session    session,
      rv_Name       name,
      rv_Name       reserved,
      char*         ledgerFile,
10  rv_Boolean      requireOldMsgs

```

Purpose

Enable an existing session for delivery tracking.

Remarks

All other `rvcm` functions require an enabled session as an argument. Programs must call `rvcm_Enable ()` before any other calls related to certified delivery.

A user can use an enabled session for both certified and non-certified communications. For example, an enabled session supports calls to `rvcm_Send ()` and `rv_Send ()`.

25 Once a session is enabled for certified delivery, it remains enabled until terminated with `rv_Term ()`. The user cannot subsequently change the certified delivery parameters of the session.

Name

```
30 rvcn_Enable ( ) promotes its session argument to a delivery-tracking session.
```

If name is NULL, then `rvcm_Enable ()` generates a unique, non-reusable name for this session.

If `name` is non-NULL, then the session binds that name. A correspondent can persist beyond session termination only when it has both a reusable name and a file-based ledger.

5 Ledger File

Every delivery-tracking session stores the state of its certified communications in a ledger, which is stored in memory associated with the process.

- 10 If `ledgerFile` is NULL, then this session uses a only process-based ledger. When the session or process terminates, all information in the ledger is lost.

- 15 If `ledgerFile` specifies a valid file name, then this session uses that file for ledger storage. If the session or process terminates with incomplete certified communications, the ledger file records that state. When a new session binds the same reusable name, it reads the ledger file and continues certified communications from the state stored in the file.

- 20 Even though a session uses a ledger file, it may sometimes replicate parts of the ledger in process-based storage for efficiency; however, programmers cannot rely on this replication.

- If the operating system supports raw storage devices (also called raw partitions), the user can specify such a device as the ledger file.

- 25 An optional prefix determines whether writing to the ledger file is a synchronous operation:

- 30 ■ To specify synchronous writing (the default), either supply an ordinary file name, or prepend the string `sync*` to the file name; for example, `"myLedger"` (implicit) or `"sync*/local/myLedger"` (explicit). Each time a ledger item is written, the call does not return until the data is safely stored in the file system.

- To specify asynchronous writing, prepend the string nosync* to the file name; for example, "nosync*/local/myLedger". The ledger file might not accurately reflect program state in cases of hardware or operating system kernel failure.

- 5 A program that uses an asynchronous ledger file can explicitly synchronize it by calling `rvcm_SyncLedgerFile()`, described below.

Parameters

10

Parameter	Description
session	Enable this session for certified delivery tracking.
name	<p>Bind this reusable name to the session, so the session represents a persistent correspondent with this name.</p> <p>If non-Null, the name must conform to the syntax rules for subject names. It cannot begin with reserved tokens. It cannot be a non-reusable name generated by another call to <code>rvcm_Enable()</code>.</p> <p>If this argument is NULL, then <code>rvcm_Enable()</code> generates a unique, non-reusable name for the duration of the session.</p>
reserved	This parameter is reserved for future enhancement. The user must supply NULL for foilNard compatibility.

Parameter	Description
ledgerFile	<p>If this argument is non-NULL, then this session uses a file-based ledger. The argument must represent a valid file name. Actual locations corresponding to relative file names conform to operating system conventions.</p> <p>Prepending nosync* to the file name specifies asynchronous output to the file system. sync* or no prefix specifies synchronous output (flushed before each output call returns).</p> <p>If this argument is NULL, then this session uses a process-based ledger.</p>
requireOldMsgs	<p>This parameter indicates whether a persistent correspondent requires delivery of messages sent to a previous session with the same name, for which delivery was not confirmed. Its value affects the behavior of other delivery-tracking senders.</p> <p>If this parameter is RV_TRUE and name is non-NULL, then this session requires certified senders to retain unacknowledged messages sent to this persistent correspondent. When this session begins listening to the appropriate subjects, the senders can complete delivery. (It is an error to supply RV_TRUE when name is NULL.)</p> <p>If this parameter is RV_FkLSr., then this session does not require certified senders to retain unacknowledged messages. Certified senders may delete those messages from their ledgers.</p>

Errors

RVCM Error Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_SESSION_NAME	The function received an ill-formed reusable name.
RVCM_ERR_NO_MEMORY	The function could not complete because the operating system denied its request to allocate storage.
RVCM_ERR_FILE_IO_ERROR	rvcm_Enable () encountered an error while opening the ledger file. For example, an explicitly named directory does not exist.
RVCM_ERR_FILE_NO_PERMISSION	File privileges are insufficient for rvcm_Enable () to open the ledger file.
RVCM_ERR_LEDGER_NAME_CONFLICT	rvcm_Enable () received NULL as the name parameter, but a non-NULL value as the ledgerFile parameter.
RVCM_ERR_PARAMETER_CONFLICT	<p>The function received conflicting values for parameters.</p> <p>rvcm_Enable () received RV_FALSE as its requireOldMsgs parameter, and NULL as its name parameter. A non-reusable name implies a transient correspondent, which cannot have backlog messages.</p>

RVCM Error Code	Indicates
RVCM_ERR_FILE_NOT_LEDGER_OWNER	The reusable name recorded in the file differs from the name of this session. <code>rvcm_Enable ()</code> stopped reading the file.
RVCM_ERR_CORRUPT_LEDGER_FILE	The ledger file is corrupt. <code>rvcm_Enable ()</code> could read only part of the ledger file into process-based memory. Some information may be lost.
RVCM_ERR_SESSION_ALREADY_ENABLED	<code>rvcm_Enable ()</code> received a session that is already enabled for delivery tracking. It is illegal to enable a session more than once.

Coding Example

```

5  cm_err = rvcm_Enable(sess, "CM_EXAMPLE", NULL,
                        "my_ledger_file", RV_TRUE);

    if (cm_err != RVCM_OK)
    {
10      fprintf(stderr, 'Can't enable CM session--%s\n',
                rvcm_ErrorText (sess, cm_err) );
        exit (-1);
    }

```

15 See Also

`rvcm_SyncLedgerFile()`, below

20 `rvcm_ListenSubject()`
Function

```
rvcm_Error rvcm_ListenSubject (
```

10 Purpose

15 **Remarks**

20

25

30

32

The software automatically confirms message delivery when the data callback function returns.

5

Parameters

Parameter	Description
session	A delivery-tracking session.
listenId	When <code>rvcm_ListenSubject()</code> returns (without error), this location contains a handle denoting the new endpoint. To stop listening on the subject, pass this handle to <code>rvcm_Close()</code> .
subject	Listen for messages with this subject name. Wildcard subjects are illegal.
dataCallbackFn	When a message arrives, pass it to this callback function.
closureArg	Pass this closure argument to the callback function. This argument must be a pointer, but it can point to any type of data. It contains any information needed by the callback function. This argument is treated as an opaque closure, forwarding it to the callback function without accessing its value.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_ListenSubject() received either a NULL callback function, or a NULL listenid pointer.
RVCM_ERR_NO_MEMORY	The function could not complete because the operating system denied its request to allocate storage.
RVCM_ERR_BAD_SUBJECT	rvcm_ListenSubject() received an ill-formed subject name. Either it is NULL, or contained too many total characters, too many characters in an element, too many elements, a wildcard character, or an illegal prefix.
RVCM_ERR_DUPLICATE_SUBJECT	rvcm_ListenSubject() Can maintain at most one open listening endpoint per subject (per session); it cannot open a second listening endpoint for this subject.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

Coding Example

```

cm_err = rvcm_ListenSubject(sess, &listenId, subject,
                           CM_callback, myClosure);
if (cm_err != RVCM_OK)
5     fprintf(stderr, "error %s listening to \"%s\"\n",
              rvcm_ErrorText (sess, cm_err), subject);

```

See Also

```
10  rvcm_Callback;  
    rvcm_ListenId;  
    rvcm_ListenInbox();  
    rvcm_Close(), all described below
```

15 **rvcm_ListenInbox()**

Function

```

20   rvcm_Error rvcm_ListenInbox (
        rv_Session      session,
        rvcm_ListenId*  listenid,
        rv_Name          inbox,
25   rv_Size             inboxLimit,
        rvcm_Callback    dataCallbackFn,
        rv_Opaque         closureArg)

```

Purpose

30 Open an inbox and begin listening for point-to-point messages addressed to it. Request certified delivery whenever available. Whenever a message arrives, pass it to the callback function.

Remarks

[illegible]

This function is parallel to *rv_ListenInbox ()*--it creates an inbox to receive point-to-point messages. The inbox receives both labeled messages and ordinary messages.

5 When a labeled message arrives from an unfamiliar delivery-tracking session, the receiving session requests certified delivery for the inbox name. If the sending session accepts the request, then the two sessions cooperate to certify delivery of subsequent messages to this inbox.

10 Certified delivery to an inbox is limited to the duration of the inbox. Once the users closes an inbox or terminates its session, the inbox and its name become obsolete; the user can never open another inbox with the same name. Since the system can never complete delivery of messages addressed to an obsolete inbox, it automatically deletes stored messages when it detects that an inbox has become obsolete.

15 The system automatically confirms message delivery when the data callback function returns.

Parameters

Parameter	Description
session	A delivery-tracking session.
listenId	When <i>rvcm_ListenInbox ()</i> returns (without error), this location contains a handle denoting the new inbox. To stop listening, pass this handle to <i>rvcm_Close ()</i> .
inbox	Location to store the generated name of the new inbox. The allocated space must be at least <i>RV_t,mX_INBOX_NAME</i> (currently 100 bytes).
inboxLimit	Number of bytes allocated to store the new inbox name.
dataCallbackFn	When a message arrives, pass it to this callback function.

Parameter	Description
closureArg	Pass this closure argument to the callback function. This argument must be a pointer, but it can point to any type of data. It contains any information needed by the callback function. the system treats this argument as an opaque closure, forwarding it to the callback function without accessing its value.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_ListenInbox () received either a NULL inbox name pointer, a NULL callback function, or a NULL listenid pointer.
RVCM_ERR_NO_MEMORY	The function could not complete because the operating system denied its request to allocate storage.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

5

Coding Example

char inboxName [RV_MAX_INBOX_NAME];

cm_err = rvcn_ListenInbox(sess, &listenId, inboxName, sizeof(inboxName),
CM_callback, myClosure);

5 if (cm_err != RVCN_OK)
fprintf(stderr, "error %s listening to certified inbox.\n",
rvcn_ErrorText(sess, cm_err));

See Also,

10 rvcn_Callback, below
rvcn_ListenId, below
rvcn_ListenSubject(), above
rvcn_Close(). below.

15 **rvcn_Callback**
Datatype

20 **Declaration**

void* rvcn_Callback (
rv_Session session,
rv_Name subject,
rv_Name replyName
25 rvmsg_Type msgType,
rvmsg_Size msgSize,
rvmsgData msg,
rvcn_Seq sequenceNum,
rv_Name senderName,
30 rv_Opaque closureArg)

Purpose

`rvcm_Callback` is the function type of data callback functions to delivery-tracking listeners. Applications define functions of this type to receive inbound messages and take appropriate actions.

5 Remarks

This function type is parallel to `rv_Callback`. Notice that its function signature includes two additional parameters, `senderName` and `sequenceNum`, to receive the tracking data that labels the inbound message. Application programs can use these arguments for testing and auditing.

10

The system automatically confirms message delivery when the data callback function returns.

15

The user can write several callback functions to process the various kinds of messages that the application expects to receive. The user can use the same callback function to process messages at several endpoints. Callback functions can take any action in processing a message, except for the following restrictions:

20

- Callback functions must not attempt to modify the data (the values of the parameters `msg`, `subject`, `reillyName` and `senderName`) in any manner. The callback function receives data through pointers; the actual data resides in memory that does not belong to the application. The data pointers remain valid only until the callback function returns.

25

- The data callback functions must return promptly.

Parameters

Parameter	Description
session	This parameter receives the current session.
subject	<p>This parameter receives the destination subject name of the inbound message.</p> <p>Do not modify this value. This pointer remains valid only until the callback function returns.</p>
replyName	<p>If the inbound message carries a name for replies, this parameter receives it. Otherwise NULL.</p> <p>Do not modify this value. This pointer remains valid only until the callback function returns.</p>
msgType	This parameter receives the datatype of the message.
msgSize	This parameter receives the size (in bytes) of the message.
msg	<p>This parameter receives a pointer to the message data.</p> <p>Do not modify this value. This pointer remains valid only until the callback function returns.</p>
sequenceNum	If the message is certified, this parameter receives the sequence number that the sender assigned to the message. Otherwise, this parameter receives zero.
senderName	<p>If the message is labeled, this parameter receives the name of the delivery-tracking session that sent the message. Otherwise, this parameter receives NULL.</p> <p>Do not modify this value. This pointer remains valid only until the callback function returns.</p>

Sequence number of a labeled message.

Remarks

- 5 Sequence numbers are limited to 32 bits on all platforms (even platforms that support 64-bit integers).

rvcm_Close()

10 *Function*

Declaration

rvcm_Error rvcm_Close (
15 rv_Session session,
rvcm_ListenId listenid

Purpose

Close a delivery-tracking endpoint; stop listening for messages on it.

20 Remarks

This function is parallel to rv_Close ().

Cooperating senders receive a REGISTRATION. CLOSED advisory, indicating that the delivery tracking agreement is no longer in effect. Senders receive a DELIVERY.

- 25 FAILED advisory for each undelivered message to the closed endpoint, and the system removes undelivered messages from the sender's ledger.

The system deletes the listener's ledger items corresponding to the closed subject.

- 30 When rvcm_Close () closes an endpoint, it generates a HOST.LISTEN.STOP advisory message to inform other applications that this application has stopped listening to the subject.

It is important that a persistent listener close a delivery-tracking endpoint when it no longer requires certified delivery of the corresponding subject—and only then. Open endpoints cause certified senders to store messages in the ledger while awaiting delivery confirmation from the sender. From the sender's perspective, a persistent listener that exits without closing its listening endpoints appears the same as a listener that terminates abnormally; the sender continues to store messages awaiting the return of the Listener process. Once an endpoint is closed, senders do not store certified messages for that listener, and successive listener processes with the same correspondent name do not receive certified delivery of messages sent in the interim.

Coding Example

Close endpoints when the application is finished receiving data. Closing endpoints lets TiB/Rendezvous software re-use the associated storage resources. After closing an endpoint, delete all references to it, to guard against closing it twice.

It is illegal to close the same endpoint more than once. On some platforms, the error `RVCM_ERR_NONEXISTENT_ID` results; on other platforms, the result can be a fatal error

(because it references dynamically allocated memory that has been freed).

This code fragment illustrates the recommended idiom for closing endpoints:

```
if (foo->listenId != NULL)
    (rvcm_Close(foo->listenId);
    foo->listenId = NULL; }
```

Parameter	Description
session	The delivery-tracking session that created the endpoint.
listenId	Close the endpoint that this handle denotes.

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_Session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_Close() received a NULL listenId pointer.
RVCM_ERR_NONEXISTENT_ID	rvcm_Close () received a NULL listenId pointer either it is NULL, or it points to something other than an rvcm_ListenId, or it points to an endpoint that has already been closed.
RVCM_ERR_NO_MEMORY	The function could not complete because the operating system denied its request to allocate storage.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

See Also

rvcm_ListenId, above.

rvcm_Send()

5 **Function**

Declaration

```
rvcm_Error rvcm_Send (  
    rv_Session      session,  
10    rv_Name       subject,  
    rvmsg_Type      msgType,  
    rvmsg_Size      msgSize,  
    rvmsg_Data      msg,  
    unsigned long   timeLimit,  
15    rvcm_Seq*     sequenceNum )
```

Purpose

Send a labeled message, and track delivery to cooperating listeners.

20 **Remarks**

This function is parallel to rv_Send (). Notice that its signature includes two additional parameters, timeLimit and SequenceNum.

Use timeLimit to specify the duration of the message (in seconds). The system retains
25 the message in its ledger until either it receives delivery confirmation from all cooperating listeners, or the timeLimit expires. If the time limit expires before delivery is complete, the system removes the message from the ledger, and generates a DELIVERY_FAILED advisory message listing the cooperating listeners that did not confirm delivery of the message.

30

We recommend a timeLimit value greater than 60 seconds, since the domain holds messages for 60 seconds.

`rvcm_Send ()` labels the message with the name of the delivery-tracking session and a sequence number. `rvcm_Send ()` passes the sequence number back to the caller in its `sequenceNum` parameter (the user can use this number for auditing).

- 5 Each sending session maintains a separate sequence for each subject it sends. As a result, receivers can uniquely identify each labeled message by the three data callback arguments `subject`, `senderName` and `sequenceNum`.

- 10 When `msg` has self-evident length, the user may supply zero for `msgSize` (the system computes the actual size automatically). Types with self-evident size are `RVMSG_RVMSG`, `RVMSG_STRING` (when the string is NULL-terminated) and `RVMSG_ENCRYPTED`. All other types require an explicit size argument in this call.

- 15 *`rvcm_Send()`*

`rvcm_Send ()` copies its arguments. After `rvcm_Send ()` returns, the user may free or reuse the storage (for example, a message buffer).

- 20 **Warning**

It is illegal to send messages to wildcard subject names. Although `rvcm_Send ()` does not prevent the user from sending illegally to wildcard subjects, the results are unpredictable.

- 25 **Parameters**

Parameter	Description
<code>session</code>	A delivery-tracking session.
<code>subject</code>	Send the message to this subject name or inbox.
<code>msgType</code>	Datatype of the message data.

msgSize	Length of the data (in bytes).
msg	Location of the data to send.
timeLimit	<p>Retain the message in the ledger, and continue attempts at delivery either until completion of delivery or until this time limit (in seconds) expires.</p> <p>Zero is a special value, indicating no time limit; the message remains in the ledger until delivery is complete to all certified listeners.</p> <p>A non-zero value less than 60 seconds adds no advantage over ordinary reliable message delivery, since rvd retains and retransmits messages for 60 seconds.</p>
sequenceNum	This location receives the sequence number of this message.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.

RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_Session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_Send () or rvcm_SendWithReply () received either a NULL subject, or a NULL sequence number pointer
RVCM_ERR_NO_MEMORY	The function could not complete because the operating system denied its request to allocate storage.
RVCM_ERR_BAD_SUBJECT	rvcm_Send () or rvcm_SendWithReply () received an ill-formed subject name. Either it contained too many total characters, too many characters in an element, or too many elements.

RVCM_ERR_SESSION_NOT_ENABLED

The function received a session that is not enabled for delivery tracking.

Coding Example

```
int time_limit = 600; /* 600sec = 10min */
```

5

```
cm_err = rvcm_Send(sess, subject, type, sizeof(msg), msg,  
                  time_limit, &seq_no);
```

```
if (cm_err == RVCM_OK)
```

```
    fprintf(stderr, "Sent sequence num %d\n", seq_no);
```

10 else

```
    fprintf(stderr, "Error while sending certified message: %s\n",  
            rvcm_ErrorText (sess, cm_err) );
```

See Also

15 rvcm_Seq, above

rvcm_SendWithReply(), below.

rvcm_SendWithReply()

Function

20

Declaration

```
rvcm_Error rvcm_SendWithReply (
```

```
    rv_Session      session,
```

```
    rv_Name;        subject,
```

25 rv_Name replyName,

```
    rvms g_Type     msgType,
```

```
    rvmsg_Size      msgSize,
```

```
    rvmsg_Data      msg,
```

```
    unsigned long timeLimit,
```

— rvcn_Seq* sequenceNum

Purpose

Send a labeled request message, and track delivery to cooperating listeners. A request message includes a reply name as a return address for response messages.

Remarks

This function is parallel to `rv_SendWithReply()`. Notice that its signature includes two additional parameters, `timeLimit` and `sequenceNum`.

10

The reply name must be the subject name of an open listening endpoint (either broadcast or inbox). An application may receive zero, one or several responses to a request message. Before the user calls `rvcn_SendWithReply()` the user must ensure that the appropriate application components are listening to the reply name.

15

`rvcn_SendWithReply()` is a non-blocking function. It returns immediately, and the callback function receives any replies asynchronously. A reply is not guaranteed (for example, if the receiver does not send any reply).

20

Use `timeLimit` to specify the duration of the message. the system retains the message in its ledger until either it receives delivery confirmation from all cooperating listeners, or the `timeLimit` expires. If the time limit expires before delivery is complete, the system removes the message from the ledger, and generates a `DELIVERY_FAILED` advisory message listing the cooperating listeners that did not

25

confirm delivery of the message.

A `timeLimit` value greater than 60 seconds is recommended, since the domain holds messages for 60 seconds.

30

`rvcn_SendWithReply()`

`rvcn_SendWithReply()` labels the message with the name of the delivery-tracking session and a sequence number. `rvcn_SendWithReply()` passes the sequence number

back to the caller in its sequenceNum parameter (the user can use this number for auditing).

5 Each sending session maintains a separate sequence for each subject it sends. As a result, receivers can uniquely identify each labeled message by the three data callback arguments subject, senderName and sequenceNum.

10 When msg has self-evident length, the user may supply zero for msgSize (the system computes the actual size automatically). Types with self-evident size are RVMSG_RVMSG, RVMSG_STRING (when the string is NULL-terminated) and RVMSG_ENNCRYPTED. All other types require an explicit size argument in this call.

15 rvcn_SendWithReply() copies its arguments. After rvcn_SendWithReply() returns, the user may free or reuse the storage (for example, a message buffer).

Warning

20 It is illegal to send messages to wildcard subject names. Although the function rvcn_SendWithReply() does not prevent the user from sending illegally to wildcard subjects, the results are unpredictable.

Parameters

Parameter	Description
Session	A delivery-tracking session.
Subject	Send the message to this subject name or inbox.

ReplyName	<p>Subject name for replies to this request message (similar to a return address on a letter, and delivered to receiving applications in the replyName argument of the callback function).</p> <p>The reply name may be an inbox or subject name. Before calling <code>rvcm_SendWithReply()</code>, ensure that a listening endpoint is already open for the reply name.</p>
MsgType	Datatype of the message data.
MsgSize	Length of the data (in bytes).
Msg	Location of the data to send.
timeLimit	<p>Retain the message in the ledger, and continue attempts at delivery either until completion of delivery or until this time limit (in seconds) expires.</p> <p>Zero is a special value, indicating no time limit: the message remains in the ledger until delivery is complete to all certified listeners.</p> <p>A non-zero value less than 60 seconds adds no advantage over ordinary reliable message delivery, since <code>rvd</code> retains and retransmits messages for 60 seconds.</p>
sequenceNum	This location receives the sequence number of this message.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid <code>rv_Session</code> (for example, <code>NULL</code> , or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument. <code>rvcm_Send()</code> or <code>rvcm_SendWithReply()</code> received either a <code>NULL</code> subject, or a <code>NULL</code> sequence number pointer.
RVCM_ERR_NO_MEMORY	The function could not complete because the operating system denied its request to allocate storage.
RVCM_ERR_BAD_SUBJECT	<code>rvcm_Send()</code> or <code>rvcm_SendWithReply()</code> received an ill-formed subject name. Either it contained too many total characters, too many characters in an element, or too many elements.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

Coding Example

```

cm_err = rvcm_SendWithReply(sess, subject, reply,
5      type, sizeof(msg), msg,
      600, &seq_no);

(cm_err == RVCM_OK)
    fprintf(stderr, "Sent sequence num %d\n", seq_no);
else
10    fprintf(stderr, "Error while sending certified message: %s\n",
        rvcm_ErrorText (sess, cm_err) );

```

See Also

rvcm_Seq, page 203; and
rvcm_SendWithReply(), above.

5

rvcm_AddListener()

Function

Declaration

10 rvcm_Error rvcm_AddListener (
 rv_Session session,
 rv_Name name,
 rv_Name subject)

15 **Purpose**

Pre-register an anticipated listener.

Remarks

20 Some sending applications can anticipate requests for certified delivery--even before the listening applications begin running. In such situations, the sender can pre-register listeners, so the system begins storing outbound messages in the sender's ledger; when the listener requests certified delivery, it receives the backlogged messages.

25 If the correspondent with this name already receives certified delivery of this subject from this sender session, then rvcm_AddListener() has no effect.

If the correspondent with this name is disallowed, then rvcm_AddListener() returns an error. The user can call rvcm_AllowListener() to supersede the effect of a prior call to
30 rvcm_DisallowListener(); then call rvcm_AddListener() again.

Parameters

RVCM_ERR_BAD_SUBJECT	The function received an ill-formed subject name. Either it is NULL, or contained too many total characters, too many characters in an element, too many elements, or an illegal prefix.
RVCM_ERR_DISALLOWED_LISTENER	rvcm_AddListener() cannot add this listener because it is disallowed. First call rvcm_AllowListener().

Coding Example

```
cm_err = rvcm_AddListener(sess, "LISTENER_17", subject);
if(cm_err != RVCM_OK)
```

```
5      {
        fprintf(stderr, "Can't add CM listener:\n %s\n",
                rvcm_ErrorText(sess, cm_err));
        exit(-1);
      }
```

10

See Also

rvcm_RemoveListener(), below.

15 **rvcm_RemoveListener()**

Function

Declaration

```
20 rvcm_Error rvcm_RemoveListener (
        rv_Session      session,
        rv_Name          name, /* listening correspondent name */
        rv_Name          subject )
```

2025

5 Remarks

10

15

20

25

58

[illegible]

5 Errors

59

RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.
RVCM_ERR_BAD_SUBJECT	The function received an ill-formed subject name. Either it is NULL, or contained too many total characters, too many characters in an element, too many elements, or an illegal prefix. rvcm_RemoveListener() received a subject containing wildcard characters; or the sender has not sent certified messages on this subject.
RVCM_ERR_DISALLOWED_LISTENER	rvcm_RemoveListener() cannot cancel certified delivery to this listener because the listener is disallowed.

Coding Example

```

cm_err = rvcm_RemoveListener(sess, listener, subject);
5  if(cm_err != RVCM_OK)
    {
        fprintf(stderr, "Can't remove CM listener:\n %s\n",
            rvcm_ErrorText(sess, cm_err));
        exit(-1);
10  }

```

See Also

rvcm_AddListener(), above; and

rvcm_DisallowListener(), below.

rvcm_DisallowListener()

5 *Function*

Declaration

rvcm_Error rvcm_DisallowListener (
 rv_Session session,
10 rv_Name name)

Purpose

15 Cancel certified delivery to all listeners at a specific correspondent. Deny subsequent
 certified delivery registration requests from those listeners.

Remarks

 Disallowed listeners still receive subsequent messages from this sender, but delivery
 is not certified. That is:

- 20
- The listener receives a REGISTRATION.NOT_CERTIFIED advisory,
 informing it that the sender has cancelled certified delivery of all subjects.
 - If the sender's ledger contains messages sent to the disallowed listener (for
25 which this listener has not confirmed delivery), then the system removes those
 ledger items, and does not attempt to redeliver those messages.
 - The system presents subsequent messages (from the cancelling sender) to the
 listener with sequence number zero, to indicate that delivery is not certified.
- 30 Senders can promptly revoke the acceptance of certified delivery by calling
 rvcm_DisallowListener() within the callback function that processes the
 REGISTRATION.REQUEST advisory.

This function disallows a correspondent by name. If the correspondent terminates, and another process instance (with the same reusable name) takes its place, the new process is still disallowed by this sender.

- 5 To supersede the effect of `rvcm_DisallowListener()`, call `rvcm_AllowListener()`.

Parameters

Parameter	Description
<code>session</code>	A delivery-tracking session
<code>name</code>	Cancel certified delivery to listeners at the session with this name.

10

Errors

RVCN_ERR Code	Indicates
<code>RVCN_OK</code>	No error. The call completed successfully.
<code>RVCN_ERR_INVALID_SESSION</code>	The function received a session argument that is not a valid <code>rv_Session</code> (for example, <code>NULL</code> or a session that has already terminated).
<code>RVCN_ERR_BAD_SESSION_NAME</code>	The function received an ill-formed reusable name.

RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_DisallowListener() received a NULL listener name.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

Coding Example

```

cm_err = rvcm_DisallowListener(sess, "LISTENER_17");
5  if(cm_err != RVCM_OK)
    {
        fprintf(stderr, "Can't disallow CM listener on %s\n",
                rvcm_ErrorText(sess, cm_err));
        exit(-1);
10  }

```

rvcm-AllowListener()

Function

15

Declaration

```

rvcm_Error rvcm-AllowListener (
    rv_Session      session,
20  rv_Name         name )

```

Purpose

Invite the named receiver to reinstate certified delivery for its listeners, superseding the effect of any previous calls to rvcm_DisallowListener().

25

Remarks

Upon receiving the invitation to reinstate certified delivery, the system at the listening program automatically sends new registration requests. The sending program accepts these requests, restoring certified delivery.

5

Parameters

Parameter	Description
session	A delivery-tracking session.
name	Accept requests for certified delivery to listeners at the session with this name.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_Session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_SESSION_NAME	The function received an ill-formed reusable name.
RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_AllowListener() received a NULL listener name.

See Also

rvcm_Callback above, and
rvcm_Confirm(), below.

5 rvcm_Confirm()

Function

Declaration

rvcm_Confirm()

10

```
    rv_Session      session,  
    rvcm_ListenId listenerId,  
    rv_Name         senderName.  
    rvcm_Seq        sequenceNumber)
```

15

Purpose

Confirm delivery of a certified message.

Remarks

20 Use this function only in programs that override automatic confirmation.

The triplet of subject name, sender and sequence number uniquely identifies each certified message. The subject name is already stored in the listening endpoint. The other two components---sender and sequence number--are explicit parameters of this

25 function.

Parameters

Parameter	Description
session	A delivery-tracking session.
listenId	Confirm delivery of a message by this listening endpoint.

senderName	Confirm delivery of a message from the sender with this correspondent name.
sequenceNumber	Confirm delivery of the message with this sequence number.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_Session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_Confirm() received a sequence number argument that is out of range.
RVCM_ERR_ID_CONFIRM_CONFLICT	rvcm_Confirm() received a listenId that automatically confirms delivery. rvcm_Confirm() is only valid after overriding automatic confirmation with rvcm_NoAutoConfirm().

RVCM_ERR_NONEXISTENT_ID	rvcm_Confirm() received an unusable listenId pointer-either NULL, or it points to something other than an rvcm_ListenId, or it has already been closed.
RVCM_ERR_NONEXISTENT_PUBLISHER	rvcm_confirm() received a senderName argument that is not recognized as the name of a certified sender.

See Also

rvcm_Callback; and rvcm_NoAutoConfirm(), above.

5 rvcm_ReviewLedger()

Function

Declaration

rvcm_Error rvcm_ReviewLedger

```

10     rv_Session      session,
        rv_Name       subject,
        rvcm_ReviewCallback reviewCallbackFn
        rv_Opaque     closureArg );

```

15 Purpose

Summarize the delivery status of messages in the ledger,

Remarks

The callback function receives one message for each matching subject stored in the ledger. For example, when rvcm_ReviewLedger() receives FOO. * as its subject

20 argument, it calls the callback function separately for these matching subjects—once for FOO.BAR, once for FOO.BAZ, and once for FOO.BOX.

However, if the callback function returns non-NULL, then `rvcm_ReviewLedger()` returns immediately.

5 If the ledger does not contain any matching items, `rvcm_ReviewLedger()` returns normally without calling the callback function.

For information about the content and format of the callback messages, see `rvcm_ReviewCallback`, below.

10 Parameters

Parameter	Description
<code>session</code>	A delivery-tracking session.
<code>subject</code>	Review ledger items with this subject. If this subject contains wildcard characters ("*" or ">"), then review all items with matching subject names. The callback function receives a separate message for each matching subject in the ledger.
<code>reviewCallbackFn</code>	This function receives the review messages.
<code>closureArg</code>	Pass this closure argument to the callback function. This argument must be a pointer, but it can point to any type of data. It contains any information needed by the callback function. the system treats this argument as an opaque closure, forwarding it to the callback function without accessing its value.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_Session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument. rvcm_ReviewLedger() received a NULL subject name or a NULL review callback function.
RVCM_ERR_NO_MEMORY	The function could not complete because the operating system denied its request to allocate storage.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

Coding Example

```
fprintf(stdout, "\nLedger review for subject '%s'. %d:\n",
        subject, reviewMax);
```

5

```
/** See rvcm_ReviewCallback example function, below. */
cm_err = rvcm_ReviewLedger (session, rv_Name)subject,
                             myReviewLedgerCallback,
                             (rv_Opaque)&reviewMax);
```

```
10 if ( cm_err != RVCM_OK)
    {
        fprintf(stderr, "Review ledger failed for '%s':\n\t%s\n",
                subject, rvcm_ErrorText(sess, cm_err));
    }
```

rvcm_ReviewCallback, above.

Datatype

```
void* rvcm_ReviewCallback (
```

15 Purpose

20 Remarks

To continue reviewing the ledger, return NULL from this callback function. To stop reviewing the ledger, return non-NULL from this callback function; `rvcm_ReviewLedger()` returns immediately.

30

Parameters

Parameter	Description
session	This parameter receives the current session.
subject	This parameter receives the subject name that the message summarizes.
msg	This parameter receives a summary message describing the delivery status of messages in the ledger. The table below describes the fields of the summary message.
closureArg	This parameter receives a closure argument, which the application supplied when it called <code>rvcm_ReviewLedger()</code> . This argument is a pointer, but it can point to any type of data. It contains any information needed by the callback function. The system treats this argument as an opaque closure, forwarding it to the callback function without accessing its value.

Message Content

Field Name	Description
subject	The subject that this message summarizes. This field has datatype <code>RVMSG_STRING</code> .
seqno_last_sent	The sequence number of the most recent message sent with this subject name. This field has datatype <code>RVMSG_UTNT</code> .

total_msgs	The total number of messages with this subject name. This field has datatype RVMSG_UINT.
total_size	<p>The total storage (in bytes) occupied by all messages with this subject name.</p> <p>If the ledger contains ten messages with this subject name, then this field sums the storage space over all of them.</p> <p>This field has datatype RVMSG_UINT.</p>
listener	<p>Each summary message can contain one or more fields named listener. Each listener field contains a nested submessage with details about a single registered listener.</p> <p>This field has datatype RVMSG_RVMSG.</p>
listener.name	<p>Within each listener submessage, the name field contains the name of the delivery-tracking listener.</p> <p>This field has datatype RVMSG_STRING.</p>
listener.last_confirmed	<p>Within each listener submessage, the last_confirmed field contains the sequence number of the last message for which this listener confirmed delivery.</p> <p>This field has datatype RVMSG_UINT.</p>

Coding Example

/** See rvcm_ReviewLedger() example call, above. */

5 void *

Synchronize the ledger file to its storage medium.

When this function returns, the session's current state is safely stored in the ledger file.

10 Delivery-tracking sessions that use synchronous ledger files need not call this function, since the current state is automatically written to the file system before returning.

Parameter	Description
session	Synchronize the ledger file associated with this delivery-tracking session.

RVCM Error Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_Session (for example, NULL, or a session that has already terminated).
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

RVCM Error Code	Indicates
	<p>rvcm_ListenSubject() received either a NULL callback function, or a NULL listenid pointer.</p> <p>rvcm_ListenInbox() received either a NULL inbox name pointer, a NULL callback function, or a NULL listenid pointer.</p> <p>rvcm_DisallowListener() received a NULL listener name.</p> <p>rvcm_AllowListener() received a NULL Listener name.</p> <p>rvcm_AdclListener() received a NULL listener name.</p> <p>rvcm_RemoveListener() received a NULL listener name or NULL subject; or the sender does not certify delivery of the subject to the listener.</p> <p>rvcm_Send() or rvcm_SendWithReply() received either a NULL subject, or a NULL sequence number pointer.</p> <p>rvcm_ReviewLedger() received a NULL subject name or a NULL review callback function.</p>

RVCM Error Code	Indicates
	NULL as its name parameter. A non-reusable name implies a transient correspondent, which cannot have backlog messages.
RVCM_ERR_FILE_IO_ERROR	<p>rvcm_Enable() encountered an error while opening the ledger file. For example, an explicitly named directory does not exist.</p> <p>rvcm_SyncLedgerFile() encountered an error while writing the ledger file.</p>
RVCM_ERR_FILE_NO_PERMISSION	File access privileges are insufficient for rvcm_Enable () to open the ledger file.
RVCM_ERR_FILE_NOT_LEDGER_OWNER	The reusable name recorded in the file differs from the name of this session. ryere_Enable () stopped reading the file.
RVCM_ERR_CORRUPT_LEDGER_FILE	<p>The ledger file is corrupt.</p> <p>rvcm_Enable() could read only part of the ledger file into process-based memory. Some information may be lost.</p>

See Also

rvcm_ErrorText(), below.

	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415	2416	2417	2418	2419	2420	2421	2422	2
--	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	---

5

10

Distributed Queue CAPI

15

Table B1: Distributed Queues: Dataty yes and Functions

Item	Description	Page
rvcm_EnableQueue()	Enable a session as a distributed queue member	248
rvcm_rvcm_SetQueueAcceptTime() 0	Set the queue time limit for task acceptance.	251
rvcm_QueueAcceptTime()	Return the queue time limit for task acceptance.	253
rvcm_SetQueueCompleteTime()	Set the queue time limit for task completion.	255
rvcm_QueueCompleteTime()	Return the queue time limit for task completion.	257

Function

```
rvcm_Error rvcm_EnableQueue (
    rv_Session      session
    rv_Name         name,
    unsigned long schedulerWeight,
    unsigned long schedulerHeartbeat,
    unsigned long schedulerActivation );
```

Enable a session for certified message delivery as part of distributed queue for one-of-n certified delivery.

83

Each member of a distributed queue listens for the same subjects—yet even when n members listen, for each inbound message (or task), exactly one member processes the message.

- 5 Programs must call `rvcm_EnableQueue()` before any other calls related to distributed queues or certified listening.

Once a session becomes a queue member, it cannot resign membership except with `rv_Term()`.

Queue Member Roles

Each distributed queue member session has two distinct roles--as a listener, and as a potential scheduler.

- 15 In the **listener role**, queue member sessions support a limited subset of certified delivery calls: `rvcm_ListenSubject()`, `rvcm_NoAutoConfirm()` and `rvcm_Confirm()`. Queue member sessions do not support any other certified delivery calls (in particular, calls associated with sending certified messages). However, they do support all standard system calls (for example, `rv_Send()`).

- 20 System fault tolerance software maintains exactly one active scheduler in each queue:
if the scheduler process terminates, another member assumes the role of scheduler.
The queue member session in the **scheduler role** assigns inbound tasks to listeners in
the queue. (A scheduler can assign tasks to its own listener component, but only does
25 so when all other listeners are busy.)

Parameters

Parameter	Description
session	<p>The call enables this session.</p> <p>Before this call, the session must not yet be enabled for delivery tracking.</p>

[illegible]

See Also

rvcm_ListenSubject(), Appendix A**rvcm_SetQueueAcceptTime0**

10 Declaration

```
rvcm_Error rvcm_SetQueueAccepCTime (
```

15

Purpose

Change the accept time parameter of a queue member session.

5 Remarks

When this session, acting as the scheduler, assigns a task to a listener (another queue member session), it sets a timer with this length (in milliseconds). If the timer elapses before the scheduler receives acceptance from the listener, the scheduler reassigns the task to another listener.

10

Zero is a special value, which specifies no limit on the acceptance time---that is, the scheduler does not set a timer, and does not reassign tasks.

15

Enabling a session as a queue member tacitly sets its accept time parameter to DEFAULT_ACCEPT_TIME (zero).

Parameters

Parameter	Description
session	A session in a distributed queue.
acceptTime	This value (in milliseconds) becomes the new time limit for acceptance of tasks. It must be less than the complete time parameter (unless the complete time is zero).

20

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.

RVCN_ERR Code	Indicates
RVCN_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_session (for example, NULL, or a session that has already terminated).
RVCN_ERR_BAD_ARG	The function received an illegal argument.
RVCN_ERR_SESSION_NOT_ENAB LED	The function received a session that is not enabled for delivery tracking.

See Also

vcm_Queue.AcceptTime(). below

5

rvcn_QueueAcceptTime()

Function

10 **Declaration**

```
#define DEFAULT_ACCEPT_TIME (0)
```

```
rvcn_Error rvcn_QueueAcceptTime (
```

```
15     rv_Session      session,
        unsigned long* acceptTime );
```

Purpose

Extract the accept time parameter of a queue member session.

20

Remarks

When this session, acting as the scheduler, assigns a task to a listener (another queue member session), it sets a timer for the accept time (in milliseconds). If the timer

09067

5

10

Parameter	Description
session	A session in a distributed queue.
acceptTime	This location receives the stored time limit (in milliseconds) for acceptance of tasks.

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.

rvcm_Qz,QueueAcceptTime()

89

Parameters

Parameter	Description
session	A session in a distributed queue.
completeTime	This value (in milliseconds) becomes the new time limit for completion of tasks. It must be greater than the accept time parameter (unless the complete time is zero).

5 Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.

RVCM_ERR Code	Indicates
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

See Also

rvcm_QueueCompleteTime(). below

5 rvcm_QueueCompleteTime()

Function

Declaration

```
#define DEFAULT_COMPLETE_TIME (0)
```

10

```
rvcm_Error rvcm_QueueCompleteTime (
```

```
    rv_Session      session.  
    unsigned long*  completeTime );
```

15

Purpose

Extract the complete time parameter of a queue member session.

Remarks

20 When this session, acting as the scheduler, assigns a task to a listener (another queue member session), it sets a timer for the complete time (in milliseconds). If the timer elapses before the scheduler receives a completion message from the listener, the scheduler reassigns the task to another listener.

25 Zero is a special value, which specifies no limit on the completion time (the scheduler does not set a timer).

Enabling a session as a queue member tacitly sets its complete time parameter to
DEFAULT_COMPLETE_TIME (zero).

30

Parameters

Parameter	Description
session	A session in a distributed queue.
completeTime	This location receives the stored time limit (in milliseconds) for completion of tasks.

5 Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.

RVCM_ERR Code	Indicates
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ABG	The function received an illegal argument.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

10 See Also

rvcm_SetQueueCompleteTime (), below

rvcm_Error rvcm_QueueCompleteTime ()

Function

5 **Declaration**

#define DEFAULT_COMPLETE_TIME (0)

rvcm_Error rvcm_QueueCompleteTime (
rv_Session session,
10 unsigned long* completeTime);

Purpose

15 Extract the complete time parameter of a queue member session.

Remarks

20 When this session, acting as the scheduler, assigns a task to a listener (another queue member session), it sets a timer for the complete time (in milliseconds). If the timer elapses before the scheduler receives a completion message from the listener, the scheduler reassigns the task to another listener.

25 Zero is a special value, which specifies no limit on the completion time (the scheduler does not set a timer).

Enabling a session as a queue member tacitly sets its complete time parameter to DEFAULT_COMPLETE_TIME (zero).

30 **Parameters**

Parameter	Description
session	A session in a distributed queue.

unsigned long listenerWeight);

Purpose

Change the listener weight parameter of a queue member session.

5

Remarks

When the scheduler receives a task, it assigns the task to the available listener with the greatest listener weight.

10

A listener is considered available unless either of these conditions are true:

- The pending tasks assigned to the listener exceed its task capacity.

15

- The listener session is the scheduler. (The scheduler assigns tasks to its own listener only when no other listeners are available.)

Enabling a session as a queue member tacitly sets its listener weight parameter to
20 DEFAULT_LISTENER_WEIGHT (1).

Parameters

Parameter	Description
session	A session in a distributed queue.
listenerWeight	This value becomes the new listener weight of the session.

Errors

5

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a <i>session</i> argument that is not a valid <i>rv_Session</i> (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

See Also

`rvcm_QueueListenerWeight()`, below.

10

`rvcm_QueueListenerWeight()`

Function

1. The first step is to identify the problem or question that needs to be addressed. This involves understanding the context and the specific requirements of the task.

```
5   rvcm_Error rvcm_QueueListenerWeight (
        rv_Session      session,
        unsigned long*   listenerWeight );
```

10

Remarks

A listener is considered available unless either of these conditions are true:

- 25

Parameters

Parameter	Description
session	A session in a distributed queue.

listenerWeight	This location receives the current listener weight of the session.
----------------	--

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a <i>session</i> argument that is not a valid <i>rv_Session</i> (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

- 5 **See Also**
 `rvcm_SetQueueListenerWeight()`, above.

`rvcm_SetQueueListenerTasks()`

10

Function

Declaration

15

```
#define DEFAULT_LISTENER_TASKS (1)
```


assign a task and signal its completion is very small compared to the time required to process the task itself. For example, when average task turnaround time is 2000 milliseconds, of which communication time contributes only 10 milliseconds to the total, then task capacity is the same as the number of processors or threads.

5

However, in some situations communication time can be significant--for example, when the queue members are distributed at distant sites connected by a Wide Area Network (WAN). When communication time is significant, the meaning of task capacity changes; instead of signifying the number of tasks that a listener can process concurrently, it signifies the number of tasks that can fill the listener's capacity despite the communication time lag. For example, when the average task turnaround time is 1500 milliseconds, of which the average task processing time contributes 1000 milliseconds to the total, then setting the task capacity to 3 minimizes the listener's idle time between tasks.

10

15

When tuning task capacity to compensate for communication time lag, balance is critical. Underloading a listener (by setting its tasks capacity too low) can cause the listener to remain idle while it waits for the schedule to assign its next task. Conversely, overloading a listener (by setting its task capacity too high) can cause some assigned tasks to wait, while other listeners that might have accepted those tasks remain idle.

20

25 Parameters

Parameter	Description
session	A session in a distributed queue.
listenerTasks	This value becomes the new listener task capacity of the session. The value must be 1 or greater.

Errors

RVCM_ERR Code	Indicates
RVCM_OK	No error. The call completed successfully.
RVCM_ERR_INVALID_SESSION	The function received a session argument that is not a valid rv_Session (for example, NULL, or a session that has already terminated).
RVCM_ERR_BAD_ARG	The function received an illegal argument.
RVCM_ERR_SESSION_NOT_ENABLED	The function received a session that is not enabled for delivery tracking.

5 See Also

rvcm_QueueListenerTasks (), below.

10

rvcm_QueueListenerTasks()

Function

15 Declaration

```
#define DEFAULT_LISTENER_WEIGHT (1)
```

```
rvcm_Error rvcm_QueueListenerTasks (
```

```

rv_Session          session,
unsigned long*      listenerTasks );

```

Purpose

5

Extract the listener tasks parameter of a queue member session.

Remarks

- 10 Task capacity is the maximum number of tasks that a listener can accept. When the number of accepted tasks reaches this maximum, the listener cannot accept additional tasks until it completes one or more of them.

- 15 When the scheduler receives a task, it assigns the task to the listener (a queue member) with the greatest listener weight--unless the pending tasks assigned to that listener exceed its task capacity. When the preferred listener has too many tasks, the scheduler assigns the new inbound task to the listener with the next greatest listener weight.

- 20 Enabling a session as a queue member tacitly sets its listener tasks parameter to DEFAULT_LISTENER_TASKS (1).

Parameters

Parameter	Description
session	A session in a distributed queue.
listenerTasks	This location receives the current listener task capacity of the session.

25

Errors

